CSC2231 First Project Progress Report
Google Calendar

Lee Chew (992234970)
Maryam Fazel-Zarandi (994849296)

1 Specification

Using a social attestation scheme we will design an interface to Google Calendar where people can publish different views of their calendar to their social networks.

2 Current Progress

We divided the project into two somewhat disjoint parts: 1) publishing new events in calendars; and 2) viewing calendars. For this progress report, we concentrated on publishing. The part on viewing will be address during the remainder of the term.

2.1 Client-Side (Lee)

I have done some research into how a GreaseMonkey script could be used to manipulate the Google Calendar page on the client side of the publisher. A script would be able to traverse the DOM tree of the page and insert an extra text box labelled "Who" for the user to specify the social groups which can view a particular event. I plan to do it both for the "Add Event" bubble (the one that pops up after clicking on a day in the main view of the calendar), as well as the "Create New Event" page – the implementation for both should be very similar.

A problem with this method is that the information entered into the "Who" text box must be sent to our server (all other information is passed to Google as usual), but the Same-Origin Policy (SOP) prevents JavaScript from sending data to a different server, or from reading or writing data belonging to a different server. Of course, there are several workarounds, some used by hackers, others used by legitimate websites like mashups. The work-around I will be implementing is to embed a 1 pixel x 1 pixel image in the Google Calendar page, and put the information typed by the user into the image URL (using the fact that when a browser requests an image and the server sends the data; the browser uploads the URL of the image.). For example, the image tag could be

<img src="/cgi-bin/saveInfo.cgi?account=LMGCal@gmail.com&event=12345&who=friends" height=1 width=1>

In this example, the publisher is LMGCal@gmail.com, the event being added has an ID of 12345 and "friends" can view the event.

The final piece of the puzzle is to actually submit the data when the user presses the save or create button after filling in the event data. GreaseMonkey scripts have the ability to register listeners to any event. Therefore I will register a listener for button -click events, and make the listener insert the image tags (which causes a query to be made to the server, and thus the information embedded in the URL to be sent to the server) when the user clicks the button. Of course, the script will have to hand off the button-click event afterwards to the existing Google code for normal processing (this is a feature of GreaseMonkey). The script will be completed by Monday.

2.2 Server-Side (Maryam)

I am using Apache 2.0 as the web server, and the MySQL database management system for storing user data. The database is composed of four tables with the following schema:

- accounts(**account_ID**, account_name, type),

- attestations(**attestation_ID**, account_ID, attestation_info),
- social_networks(**sn_ID**, sn_name, account_ID),
- events(**event_ID**, event_name, sn_ID).

In the above, fully underlined words represent primary keys, and the dashed ones represent foreign keys. Type in the accounts table can be either "publisher" or "viewer". Viewers would have to upload their attestations which will be stored in the attestations table. The events added by publishers, as well as the specified social network(s) which can view the event will be stored in the events and social_networks table respectively.

If a publisher wants to use our service, he/she must go to LM.com and add their name along with their GMail username in the form provided. The server stores this data (with type=publisher) in the accounts table. In order for our service to access a publisher's calendars, the publisher would also have to add our service's GMail account (for example, LMGCal@gmail.com) as an owner of the calendars he/she wants to be viewed by others.

I use Perl and cgi-lib.pl (the standard library for creating Common Gateway Interface (CGI) scripts in the Perl language) to processes the data entered by the user as sent to the server through the <img> tag in the fashion discussed in the previous section. The script basically parses the URL plus the parameters sent to it and stores the data in the appropriate MySQL tables. The code can be found in Appendix I at the end of this document (saveInfo.cgi).

3 Future Work

The part that remains deals with viewing published calendars and is briefly summarized in the remaining of this section. The timetable for the remaining of the project is as follows:

- 2 weeks for working with social attestations
- 2 weeks for creating the viewer Firefox plugin and implementing calendar retrieval and display (summarized in the next sections)
- 1 week for testing and debugging
- 1 week for writing the final report

3.1 Viewer Firefox Plugin

This will be a browser plugin that provides the user with the following features:

1. An upload button, which will allow the user to select attestations to upload to our server. Pressing the button will actually bring up a regular File Chooser dialog box.

2. A text box to type in the email address of the publisher whose calendars he/she would like to view (i.e. similar to how a user might type search terms into a Google toolbar).

3. The ability to answer the challenge sent by the server, or in other words verify the viewer to our server. Since we will be using a public-private key scheme, the plugin will need to be able to decode challenges using the user's private key. The private key will be specified during installation and there will be an option to change it afterwards.

3.2 Calendar Retrieval and Display

We will be using the event data stored in the database (when they where created by the publisher) along with the Google Calendar API for this. In order for our service to access a publisher's calendars, he/she would have to add the service as an owner of the calendars he/she wants to be viewed by others. Upon a view request, we can use access-control-lists kept by Google to retrieve calendars belonging to a specified publisher, and using the data we have collected, we can query for the events that can be viewed by the viewer.

4 Appendix I – saveInfo.cgi

```
#!/perl/bin/perl
require "cgi-lib.pl";
use DBI;

&ReadParse(*input);

$account = $input{'account'};
$event = $input{'event'};
$who = $input{'who'};
@who = split(' ',$who);

$db_handle = DBI->connect(
          "dbi:mysql:database=osn;host=localhost:3306;user=root;password=****")
          or die "Couldn't connect to database: $DBI::errstr\n";

if (!defined $who) {
   for $i (0 .. $#who) {

          $sql = "SELECT * FROM social_networks WHERE account='$account' AND
                 sn_name='$who[$i]'";
          $statement = $db_handle->prepare($sql) or
                        die "Couldn't prepare query '$sql': $DBI::errstr\n";
          $statement->execute() or
                 die "Couldn't execute query '$sql': $DBI::errstr\n";

          if ($row_ref = $statement->fetchrow_hashref()){}
          else{
                 $sql = "INSERT INTO social_networks (account, sn_name)
                        VALUES ('$account', '$who[$i]')";
                 $statement = $db_handle->prepare($sql) or
                        die "Couldn't prepare query '$sql': $DBI::errstr\n";
                 $statement->execute() or
                        die "Couldn't execute query '$sql': $DBI::errstr\n";
          }

          $sql = "SELECT * FROM social_networks WHERE account='$account' AND
                  sn_name='$who[$i]'";
          $statement = $db_handle->prepare($sql) or
                        die "Couldn't prepare query '$sql': $DBI::errstr\n";
          $statement->execute() or
                 die "Couldn't execute query '$sql': $DBI::errstr\n";
          $foreign = $row_ref->{sn_ID};

          $sql = "SELECT * FROM events WHERE event_name='$event' AND
                  sn_ID='$foreign'";
          $statement = $db_handle->prepare($sql) or
                        die "Couldn't prepare query '$sql': $DBI::errstr\n";
          $statement->execute() or
                        die "Couldn't execute query '$sql': $DBI::errstr\n";
          if ($row_ref = $statement->fetchrow_hashref()){
                 print "Event exits!!!</b>.<br>\n";
          }
          else{
                 $sql = "INSERT INTO events (event_name, sn_ID) VALUES
                        ('$event', '$foreign')";
                 $statement = $db_handle->prepare($sql) or
                        die "Couldn't prepare query '$sql': $DBI::errstr\n";
                 $statement->execute() or
                        die "Couldn't execute query '$sql': $DBI::errstr\n";
          }
   }
}

$db_handle->disconnect();
```